Compiler-Directed Power Density Reduction in NoC-Based Multi-Core Designs

Sri Hari Krishna Narayanan, Mahmut Kandemir, Ozcan Ozturk
Department of Computer Science and Engineering
The Pennsylvania State University
{snarayan,kandemir,ozturk}@cse.psu.edu

Abstract

As transistor counts keep increasing and clock frequencies rise, high power consumption is becoming one of the most important obstacles, preventing further scaling and performance improvements. While high power consumption brings many problems with it, high power density and thermal hotspots are maybe two of the most important ones. Current architectures provide several circuit based solutions to cope with thermal emergencies when they occur but exercising them frequently can lead to significant performance losses. This paper proposes a compiler-based approach that balances the computational workload across the processors of an NoC based chip multiprocessor such that the chances of experiencing a thermal emergency at runtime are reduced. Our results show that the proposed approach cuts the number of runtime thermal emergencies by 42% on the average on benchmarks tested.

©2006 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

This paper was supported by NSF Career Award 0093082, NSF grant 0202007 and a grant from the GSRC.

Compiler-Directed Power Density Reduction in NoC-Based Multi-Core Designs*

Sri Hari Krishna Narayanan, Mahmut Kandemir, Ozcan Ozturk
Department of Computer Science and Engineering
The Pennsylvania State University
{snarayan,kandemir,ozturk}@cse.psu.edu

Abstract

As transistor counts keep increasing and clock frequencies rise, high power consumption is becoming one of the most important obstacles, preventing further scaling and performance improvements. While high power consumption brings many problems with it, high power density and thermal hotspots are maybe two of the most important ones. Current architectures provide several circuit based solutions to cope with thermal emergencies when they occur but exercising them frequently can lead to significant performance losses. This paper proposes a compiler-based approach that balances the computational workload across the processors of an NoC based chip multiprocessor such that the chances of experiencing a thermal emergency at runtime are reduced. Our results show that the proposed approach cuts the number of runtime thermal emergencies by 42% on the average on benchmarks tested.

1 Introduction

High power dissipation is a very serious issue in today's microprocessors, due to increasing transistor counts and clock frequencies. One of the problems related to high power consumption is thermal emergencies [3, 10], which can be defined as the execution state at runtime where a certain temperature threshold is reached and current computation cannot continue as it is. If not tackled appropriately, thermal emergencies can lead to disastrous scenarios in terms of performance degradation and equipment loss.

Prior solutions to controlling thermal emergencies include circuit and architectural techniques such as [7, 10]. As a practical example, emergency overheating detector is a circuit technique that first appeared in the P6 series, which is also implemented in the Pentium 4, Xeon and Pentium M processors. In this technique, when a certain thermal threshold is reached, the processor suspends its execution (or powers off). While this and other similar techniques can prevent chip burn-outs, frequent suspensions of execution is costly in terms of performance. This is particularly problematic in real-time systems. Also, in multiprocessor systems, a small set of processors can easily become hotspots as a result of the workload imbalance across the processors.

This paper proposes and experimentally evaluates a compiler-directed scheme that balances the computational work across the processors in an NoC based chip multiprocessor, where processors are organized as a two-dimensional mesh. The proposed compiler-directed approach makes use of ILP (integer linear programming) and operates in two phases. In the first phase, it determines the largest mesh area that can be occupied by the parallel computation without exceeding the performance degradation tolerance specified. In the second phase, it splits the workloads of select processors across multiple mesh nodes to further eliminate potential hotspots by balancing out the power density.

We implemented this approach and tested its behavior using a set of five applications. Our experiments reveal two important results. First, the scheme works well to reduce the occurrences of thermal emergencies. Second, the scheme also works well to improve the performance.

The rest of this paper is organized as follows. Section 2 discusses related work and Section 3 presents the high level view of our approach. Section 4 gives details of the ILP approach, Section 5 presents our runtime results and Section 6 concludes the paper.

2 Related Work

Thermal hotspots have been identified as an important design concern in modern processors [2, 5, 9, 11, 12]. There exist solutions to this problem based on runtime techniques involving additional hardware. Usually temperature sensors are placed in the chip, which report the temperature to an arbiter, which in turn decides whether a particular block is too hot to continue operation or whether communication needs to be rerouted away from a potential hotspot.

Works such as [13] perform static temperature aware scheduling such that no processor in a CMP (chip multipro-

*This paper was supported by NSF Career Award 0093082, NSF grant 0202007 and a grant from the GSRC.

cessor) rises above the safe working threshold temperature. While [13] focusses on a bus-based system, out work targets NoC based architectures.

Activity migration proposed in [7] as a runtime technique reduces temperature in CMP environment by ping-ponging jobs on multiple processing elements. This method migrates computation to a different part of the chip if the temperature of one processing unit goes past a certain limit. This study does not consider the NoC design where the buffers and routers themselves can also become hot.

A runtime technique for managing thermal emergencies in the context NoCs has been presented in [10]. It proposes a throttling mechanism in order to route communication away from a potential thermal hot spot. The work described in this paper is different from these prior efforts in that it does not propose changing the dynamic runtime mechanisms in the hardware that prevent chip damage. Instead, it proposes a compiler directed task mapping mechanism that reduces the number of such thermal emergency occurrences at runtime. Consequently the proposed method can work in tandem with any runtime scheme in question.

3 Overall Approach

In the context of NoCs, a runtime temperature control mechanism should not allow a processor or router to function normally if it approaches a temperature threshold τ.

Hence, it is important to reduce the number of such occurrences of thermal emergencies that cause shut downs. This is possible by reducing the power density otherwise known as the power per unit area of the chip. Normally, the nature of a default (i.e., performance oriented) mapping of tasks to processors is such that the communication cost between pairs of communicating processors is reduced. This however has the inverse effect of increasing the power density in a certain area of the chip (where the application is mapped), which causes performance to deteriorate (when a thermal emergency is experienced). The approach proposed in this paper is to reduce the power density of the entire active area of the chip, which is defined as the bounding rectangle formed by the active processors on the chip.

The power density can be reduced in at least two ways in the context of NoCs. First, the overall power density is reduced by increasing the size of the bounding rectangle, i.e., the rectangle in the mesh space to which the application is mapped. Second, the power density at any point within a bounding box is reduced by splitting a task scheduled originally to be executed on one processor across multiple processors. In this work, we assume that the underlying network architecture is exposed to the compiler and that the compiler can specify the task-to-processor mapping to reduce the power density.

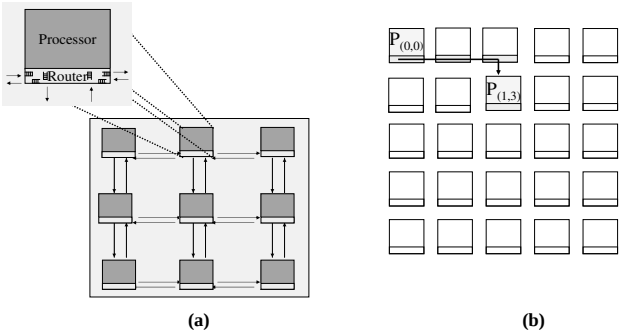

Figure 1. (a) High level view of a 3*3 NoC architecture. (b) Example of X-Y routing.

4 Mathematical Programming Model

4.1 NoC Architecture

NoC architectures [1, 4, 8] have been proposed to overcome the problems associated with long wires used in chip wide communication. They allow a regular means of communication between on-chip computation blocks, eliminate unwanted timing complexities, and increase the bandwidth and latency of communication [4]. Figure 1(a) shows the high level architecture of the NoC-based multiprocessor considered in this paper. It comprises of a grid of processors, with a router for each processor. There exists a physical, wired connection from router to router.

The rest of this section presents the main constraints of the ILP (integer linear programming) based model proposed. Due to space constraints, the actual linear constraints are not presented; instead, their non-linear equivalents are presented. Recall that our main goal is to modify a given default (performance oriented) task-to-processor mapping to reduce the number of thermal emergencies.

The high level view of the implementation of our approach is shown in Figure 2. The original application is mapped onto a set of processors by the application mapping module which is provided as input to our module, which consists of two phases. In the first phase, the given task to processor mapping is modified so that the area (bounding box) occupied by active processors is increased. In the second phase, tasks that expend too much power on one processor are split and distributed among multiple processors and a new thermal aware mapping within the area calculated in the first phase is found.

4.2 Phase 1

The input to our approach is a performance oriented task-to-processor mapping as shown in Figure 2. This mapping is captured by the matrix A in our ILP formulation. It is

Table 1. Terms used in the formulation.

Term	Explanation
$A_{i,j}$	Original task-to-processor mapping
$B_{i,j}$	Task-to-processor mapping generated by Phase 1
$C_{i,j,k,l}$	Original communication between $A_{i,j}$ and $A_{k,l}$
$C'_{i,jk,l}$	Communication between $B_{i,j}$ and $B_{k,l}$
$D_{i,j}$	Task-to-processor mapping generated by Phase 2
ρ	Power threshold allowed on a processor
$\mathcal{P}_{i,j}$	Input power utilization array
$R_{X_{i,j}}$	Router utilization due to horizontal communication
$R_{Y_{i,j}}$	Router utilization due to vertical communication
$R_{XY_{i,j}}$	Router utilization as a corner router
$\mathcal{T}_{i,j,k,l}$	Processor-to-processor mapping between $A_{i,j}$ and $B_{k,l}$
$\mathcal{T}'_{i,j,k,l}$	Processor-to-processor mapping between $B_{i,j}$ and $D_{k,l}$
θ	Router power
τ	Threshold temperature
W_{energy}	Total window energy
$Area$	Bounding rectangle formed by active processors
$Relaxation$	Extra communication allowance given
$\mathcal{W}$	Window size

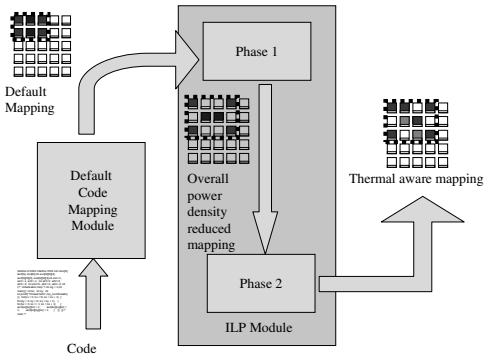

Figure 2. High level view of our scheme.

assumed that processors communicate using X-Y routing. Figure 1(b) shows an example X-Y routing for communication between processors $A_{(0,0)}$ and $A_{(1,3)}$.

Number of Active processors constraints.

Constraint 1 below specifies that the number of active processors and the original task-to-processor mapping, A, and the new mapping, B, is the same.

$$\sum_{i=1}^{n}\sum_{j=1}^{m} B_{i,j} = \sum_{i=1}^{n}\sum_{j=1}^{m} A_{i,j} \quad (1)$$

Graph embedding constraints.

Constraint 2 below states that the variable $\mathcal{T}_{i,j,k,l}$ is 1 if the task originally assigned to processor $A_{i,j}$ is allocated to $B_{k,l}$ in the new mapping. Constraint 3 ensures that each processor in B executes at most one task. Constraint 4 states that the number of processor-to-processor mappings between A and B is equal to the number of active processors in A.

$$\forall_{i,j} \quad \sum_{k,l} \mathcal{T}_{i,j,k,l} \geq A_{i,j} * B_{k,l} \quad (2)$$

$$\forall_{k,l} \quad \sum_{i,j} \mathcal{T}_{i,j,k,l} \leq 1 \quad (3)$$

$$\sum_{i,j,k,l,} \mathcal{T}_{i,j,k,l} = \sum_{i=1}^{n}\sum_{j=1}^{m} A_{i,j} \quad (4)$$

Communication constraints.

Constraints 5, 6, 7, 8 and 9 given below capture the total utilization of a router due to communication among processors. $C'_{r1,c1,r2,c2}$ is a binary matrix entry that takes the value 1 if processor $P_{r1,c1}$ communicates with processor $P_{r2,c2}$ (in the new mapping). $R_{X_{i,j}}$ and $R_{Y_{i,j}}$ give the total number of pairs of processors that use the router $R_{i,j}$ for communication in the X (horizontal) and Y (vertical) direction respectively. On the other hand, $R_{XY_{i,j}}$ counts the number of pairs of processors that use the router $R_{i,j}$ for communication in the X and the Y direction. Constraint 9 specifies that Sum_R is must be less than the original communication cost plus the relaxation allowed. [1]

$$\forall_{i,j} \quad R_{X_{i,j}} = \sum_{\substack{r1,c1,r2,c2|i=r1\&\& \\ min(c1,c2)\leq j\leq max(c1,c2)}} C'_{\substack{r1,c1, \\ r2,c2}} \quad (5)$$

$$\forall_{i,j} \quad R_{Y_{i,j}} = \sum_{\substack{r1,c1,r2,c2|j=c2\&\& \\ min(r1,r2)\leq i\leq max(r1,r2)}} C'_{\substack{r1,c1, \\ r2,c2}} \quad (6)$$

$$\forall_{i,j} R_{XY_{i,j}} = \sum_{\substack{r1,c1,r2,c2|i=r1\&\& \\ j=c2}} C'_{r1,c1,r2,c2} \quad (7)$$

$$Sum\_R = \sum_{i,j} R_{X_{i,j}} + R_{Y_{i,j}} - R_{XY_{i,j}} \quad (8)$$

$$Sum\_R \leq Input\_R + Relaxation \quad (9)$$

Constraints 10 and 11 given below state that two processors can communicate only if they are both active. Constraints 12 and 13 state that a communication between two processors in the new task-to-processor mapping (B) exists if two processors in the original task-to-processor mapping (A) map on the processor in the new mapping and the original processors themselves communicated.

$$\forall_{i,j,k,l} C'_{i,j,k,l} \leq \mathcal{A}_{i,j} \quad (10)$$

$$\forall_{i,j,k,l} C'_{i,j,k,l} \leq \mathcal{A}_{k,l} \quad (11)$$

$$\forall_{\substack{i,j,k,l, \\ m,n,o,p}} \mathcal{T}_{\mathcal{C}'\ \substack{i,j,k,l, \\ m,n,o,p}} = \mathcal{T}_{i,j,m,n} * \mathcal{T}_{k,l,o,p} \quad (12)$$

$$\forall_{\substack{i,j,k,l, \\ m,n,o,p}} C'_{m,n,o,p} = \mathcal{T}_{\mathcal{C}'\ \substack{i,j,k,l, \\ m,n,o,p}} * C_{i,j,k,l} \quad (13)$$

[1] The allowable performance degradation in this work is specified as the amount of extra communication latency permitted.

Area constraint and objective function.

Constraint 14 specifies that the area of the chip occupied by the active processors. The objective funtion 15 maximizes the area of the chip. Note that this can increase the communication cost. However, the cost has to be kept under the sum of the old communication cost and the relaxation allowed (captured by the variable $Relaxation$). Therefore, this model works towards increasing the area while keeping the communication costs under a specified limit.

$$Area = \big(max(Row) - min(Row)\big) * \big(max(Col) - min(Col)\big) \quad (14)$$

$$maximize(Area) \quad (15)$$

4.3 Phase 2

The second phase of the ILP formulation accepts as input the task-to-processor mapping B, and the communication map, C', given by Phase 1. This second phase now considers each processor as being unique based on the power it expends, which is related to the nature of the task assigned to that processor. It then splits tasks that consume too much power among multiple processors. The nature of the split is such that the area calculated by Phase 1 forms the upper bound of the area usable by Phase 2.

It needs to be noted that the very nature of splitting the computation means that the number of processors that communicate with each other increases. It is possible that the communication characteristics of this new sub-computation is different from the original computation; however, in this paper, a conservative approach is adopted and all communication is assumed to be uniform. That is, if two processors, $P_{i,j}$ and $P_{k,l}$, communicate in mapping generated by phase one and if in the new mapping $P_{i,j}$ is split into $P_{m,n}$ and $P_{o,p}$; then, in the new mapping, it is assumed that $P_{m,n}$ and $P_{o,p}$ communicate with $P_{k,l}$ and no qualification about the communication is made.

Increased active processors constraint.

The splitting of the tasks that consume power above a threshold, τ, is achieved by Constraint 16. Here, $P_{i,j}$ is the power consumption of a processor in $B_{i,j}$. The new mapping is given by $D_{i,j}$ and the threshold power value used in splitting is ρ.

$$\sum_{i=1}^{n}\sum_{j=1}^{m} D_{i,j} = \sum_{i=1}^{n}\sum_{j=1}^{m}(\mathcal{P}_{i,j}/\rho) \quad (16)$$

Embedding constraints.

The mapping from processors in B to those in D is captured by constraint 17. That is ${\mathcal{T}'}_{i,j,k,l}$ is 1 if a task executing on a processor $B_{i,j}$ maps onto $D_{k,l}$. Constraint 18 ensures that a processor in D runs only one task of B. The mapping of split processors is accounted for by constraint 19. According to this constraint, a task mapped onto processor in B, if split into multiple tasks, will cause that particular processor in B to be linked as many processors in D as the number of split tasks.

$$\forall_{i,j} \sum_{k,l} {\mathcal{T}'}_{i,j,k,l} \geq B_{i,j} * D_{k,l} \quad (17)$$

$$\forall_{k,l} \sum_{i,j} {\mathcal{T}'}_{i,j,k,l} \leq 1 \quad (18)$$

$$\forall_{i,j} \sum_{k,l} {\mathcal{T}'}_{i,j,k,l} = \sum_{i=1}^{n}\sum_{j=1}^{m}(\mathcal{P}_{i,j}/\rho) \quad (19)$$

Communication constraints.

These are similar to constraints 5 to 13 and, thus, are not presented again.

Area constraint.

This is similar to constraint 14 and is not presented again.

Energy window constraint.

It is ensured by constraint 20 below that power dissipation is spread within the active area of the chip. This is done by ensuring that, for any square window, of size $W * W$, within the chip, the total power dissipation due to processors and routers is within a preset limit W_{energy}. This has a sort of flattening effect on the power consumption withing the active area, as the area in which power is dissipated is reduced and activity is evenly spread across the active area. Here, ρ and θ are the threshold consumption of power and the power spent by one pair of communicating processors in a router, respectively.

$$\forall_{i\in 0..R-W+1, j\in 0..C-W+1} \sum_{r=i}^{i+W-1}\sum_{c=j}^{j+W-1} \big(D(r,c) * \tau + R'(r,c) * \theta\big) \leq W_{energy} \quad (20)$$

Objective function.

The objective function in this phase of our ILP formulation is to minimize the total communication cost, which ensures that the overall power density is lowered as well. This is because the total power depends on the amount of processing done (which remains constant) and the amount of communication. Since constraint 21 below reduces the communication, it helps to reduce the power and hence power density as well.

$$minimize(R') \quad (21)$$

Algorithm 1

```
1: //Time_Taken calculates the total execution time
2: Time_Taken := 0
3: while all chunks on all processors are not scheduled do
4:    Time_Taken := Time_Taken + 1
5:    T_{i+δ} = HS(T_i, floorplan, power, schedulable, cycles, δ)
6:    //the time is incremented and HotSpot function is called to estimate the
      temperature
7:    for each processor p do
8:       //Schedulable(p) indicates whether processor p is active
9:       //Chunk_Count(p) is the amount of computation p needs to perform
10:      if (Schedulable(p) =1) then
11:         Chunk_Count(p) := Chunk_Count(p) -1
12:      end if
13:      //If p is too hot, it cannot be active
14:      if Temperature(p) > τ then
15:         Schedulable(p) = 0
16:      else
17:         Schedulable(p) = 1
18:      end if
19:   end for
20:   //If the router is too hot, processors communicating via it, are switched
      off
21:   for each router r do
22:      if Temperature(r) > Threshold then
23:         for all pairs of processors (p1,p2) communicating via r do
24:            Schedulable(p1) = 0
25:            Schedulable(p2) = 0
26:         end for
27:      end if
28:   end for
29:   //Estimate the amount of communication taking place via each router r
30:   for all routers r do
31:      Schedulable(r) =0
32:   end for
33:   for all routers r do
34:      for all pairs of processors (p1,p2) communicating via r do
35:         if Schedulable(p1) = 1 && Schedulable(p2) =1) then
36:            Schedulable(r) := Schedulable(r) + 1
37:         end if
38:      end for
39:   end for
40: end while
```

4.4 Implementation Details

Our implementation in this work uses the HotSpot tool [12] in order to estimate the temperature of on-chip elements. HotSpot takes as input the floorplan of the chip, the temperature at any time i (T_i) of each element of the chip, and the power consumption of that element during a time period δ and returns the temperature of each element at time $i+\delta$. In mathematical terms, this can be represented as:

$$T_{i+\delta} \quad = \quad HS(T_i, floorplan, power, cycles, \delta)$$

In order to simulate the task-to-processor mapping, runtime processor activity and runtime processor shutdown, the algorithm shown in Algorithm 1 is used. The tasks are broken into sub-tasks called chunks. The scheduling is at the granularity of chunks. The algorithm calculates the temperature at each step and, if the temperature of a router approaches the threshold, the concerned processor is shutdown. If a router becomes too hot, then all processors that communicate using that processor are shut down. The time at which the last task completes is taken as time of completion of the particular mapping.

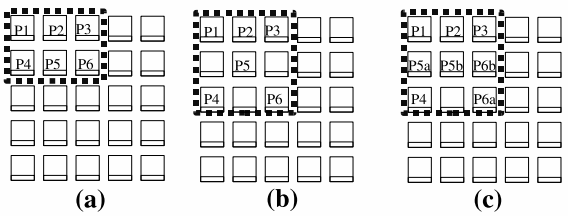

Figure 3. (a) Default mapping. (b) Mapping produced by the first phase. (c) Mapping produced by the second phase.

Table 2. Benchmarks used.

Benchmark	Cycles Millions	Processor Energy (μJ)	Router Energy (μJ)
adi	438	1239551.1	604697
eflux	56	80918.1	1696502
tsf	1799	2548001.6	515800
syntc1	438	1239551.1	0
syntc1	56	80918.1	85917071

4.5 Example Application of Our Approach

Figure 3 shows an example application of our scheme. The default (performance-oriented) mapping is given in Figure 3(a). The grid size is $5 * 5$. There are 6 tasks and they all communicate with each other. Tasks 1, 2, 3, and 4 are assumed to have power level 1. Tasks 5 and 6 are assumed to have power level 2. Hence, the average power density is $8/6 = 1.25$ W/unit area. The threshold power level, ρ, is assumed to be 2. Figure 3 shows the mapping obtained at the end of the first phase of our scheme. As can be seen, the active area of the processors has increased from 6 to 9, which causes the average power density to reduce ($8/9 = .88W/unitarea$); but the power density of processors 5 and 6 is still 2. In phase 2, the workloads of processors 5 and 6 are split into 2 sub-tasks each (5_a, 5_b, 6_a, 6_b) of power level 1. This causes the average power density to remain the same, but power density at any point now is at most 1.

Table 3. Architectural details.

Parameter	Brief Explanation
Processor	300MHz single issue
Chip Area	8.2mm * 7mm
τ	86.12 $^\circ C$
W	1
Mesh size	5 * 5 Grid
Processor Area	1.4mm * 1.4 mm
Router Area	.24mm * 1.4mm

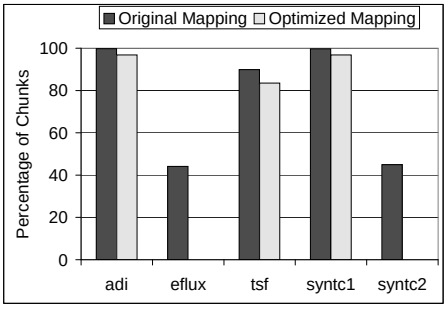

Figure 4. Percentage of execution chunks in which a thermal emergency is dealt with.

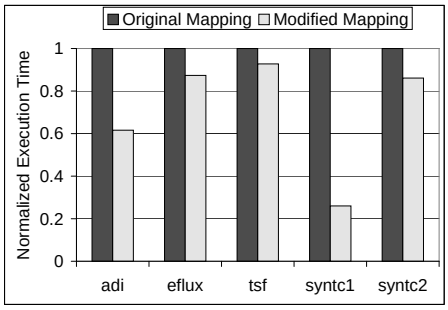

Figure 5. Normalized performance of the original and optimized mappings.

5 Experimental Results

Five loop-intensive parallel algorithms with characteristics given in Table 2 are used as benchmarks. The second column of this table gives the cycles for execution for each of the benchmarks and the third and fourth columns give the processor and router energy numbers, respectively, under the performance oriented mapping. The last two benchmarks are synthetic benchmarks that exhibit different extreme communication patterns. Specifically, in $syntc1$, very little inter-processor communication takes place, whereas in $syntc2$, the processors communicate frequently. The ILP solver used is XPressMP [6]. The details of the architecture simulated are given in Table 3.

Figure 4 shows, for each benchmark, the percentage of chunks in which a thermal emergency that requires the runtime mechanism to intervene is seen. Note that these numbers only capture the absolute number of chunks in which an emergency is seen, the number of elements affected by such an emergency is captured by the overall performance plot given below. We see from Figure 4 that our thermal-aware approach cuts the the number of thermal emergencies by 42% on average.

Figure 5 shows the normalized performance of each of the benchmarks for the default mapping and the optimized mapping. As can be seen from this bar-chart, the proposed method reduces the overall execution time by 29% on average. This large reduction is a result of the reduction in the number of times a thermal emergency occurs. The improved performance values also capture the number of thermal emergencies that that occur in a chunk in which a thermal emergency is noted.

6 Conclusion

This paper proposes a novel compiler-directed scheme to reduce the occurrences of thermal emergencies in NoC based chip multi-processor systems. The proposed scheme reduces the power density in the active area of the chip, which is the primary cause of runtime thermal emergencies. The scheme reduces the occurrence of thermal emergencies in all benchmarks tested and improves the overall performance as well. The proposed scheme is implemented using ILP and works orthogonally with all dynamic, hardware based schemes that handle runtime emergencies.

References

[1] L. Benini and G. deMicheli. Networks on chips: A new SoC paradigm. IEEE Comp., 35(1), 2002. *IEEE Comp., 35(1)*, 2002.

[2] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In *Proc. of the International Symposium on High-Performance Computer Architecture,* Jan. 2001.

[3] P. Chaparro, G. Magklis, J. Gonzlez and A. Gonzlez. Distributing the Frontend for Temperature Reduction In *Proc. of the International Symposium on High-Performance Computer Architecture,* 2005.

[4] William J. Dally, Brian Towles Route packets, not wires: on-chip interconnection networks In *Proceedings of the Design Automation Conference* , 2001.

[5] J. Donald and M. Martonosi. Temperature-Aware Design Issues for SMT and CMP Architectures. In *Proc. of the Workshop on Complexity-Effective Design* , June 2004.

[6] C. Guret, C. Prins and M. Sevaux. Applications of optimization with Xpress-MP. *Dash Optimization*, 2002.

[7] S. Heo, K. Barr, and K. Asanovic. Reducing Power Density through Activity Migration. In *Proceedings of the 2003 International symposium on Low power electronics and design.*2003.

[8] J. Hu, R. Marculescu. Energy-Aware Mapping for Tile-based NOC Architectures Under Performance Constraints, In *Proc. Asia South Pacific - Design Automation Conference*, 2003.

[9] E. Rohou and M. Smith. Dynamically managing processor temperature and power. In *Proc. of the Second Workshop on Feedback-Directed Optimization*, Nov. 1999.

[10] L. Shang, L. S. Peh, A. Kumar, N. K. Jha, Thermal Modeling, Characterization and Management of On-Chip Networks, In *Proc. of the International Symposium on Microarchitecture* , 2004.

[11] K. Skadron, T. Abdelzaher, and M. R. Stan. Control-Theoretic Techniques and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In *Proc. of the 2002 International Symposium on High-Performance Computer Architecture,* February, 2002.

[12] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-Aware Microarchitecture. In *Proceedings of the 30th International Symposium on Computer Architecture,* pp. 2-13, June 2003.

[13] N. Sri Hari Krishna, G. Chen, M. Kandemir, Y. Xie. Temperature-Sensitive Loop Parallelization for Chip Multiprocessors. In *Proceedings of the International Conference of Computer Design*, 2005.